Multipoint Session Types for a Distributed Calculus

Eduardo Bonelli[1] and Adriana Compagnoni[2]

[1] LIFIA, Fac. de Informática, UNLP, Argentina and CONICET
eduardo@lifia.info.unlp.edu.ar
[2] Stevens Institute of Technology, Hoboken NJ 07030, USA
abc@cs.stevens.edu

Abstract. Session types are a means of statically encoding patterns of interaction between two communicating parties. This paper explores a distributed calculus with session types in which a number of fixed sites interact. The reduction schemes describing the operational semantics satisfy the locality principle: at most one site is involved. Both session engagement and data communication are local and asynchronous. Furthermore, our setting is a natural one in which the novel notion of multipoint session types, sessions in which more than two parties may be involved, can be introduced.

1 Introduction

We study a type based approach to structuring interaction between multiple distributed parties. A natural way of specifying interactions is to describe them in terms of sequences of types of the entities being sent or received. This is the idea behind *session types* [Hon93, HKT94, HVK98]. We develop a theory of session types for a core distributed calculus called DCMS (*distributed calculus with multipoint session types*). Regarding the distributed nature of DCMS we take, as fundamental working hypothesis, that the schemes defining its semantics follow the locality principle [Bou03]: all such schemes should involve at most one site.

In DCMS a *site* is an expression of the form $n[\![e]\!]$ where n is the name of the site and e is a *thread expression*. In order for sites to communicate we assume they share some set of global names which we refer to as *ports* given their similarity in nature to TCP/IP port numbers. Before exchanging information, however, sites must first establish a private channel through such a port. In all extant calculi with session types this is achieved via some variation of the following reduction scheme [HVK98]:

$$\mathsf{request}\, a(k:s)\,\{P\} \,|\, \mathsf{accept}\, a(k:\overline{s})\,\{Q\} \longrightarrow (\nu k)(P \,|\, Q)$$

Here a is the aforementioned port and s a *session type* indicating the communication pattern to be followed on the fresh private channel k. For example, s

could be $\mathsf{!int.!int.?int}.end$ if the process to the left of the pipe were connecting to an adding server (the process to the right) that receives two numbers and sends back their sum. The output type $\mathsf{!int}$ is read as "send an int" and the input type $\mathsf{?int}$ as "receive an int". The session type $\overline{s}$ is the *dual* of s, in this case reading $\mathsf{?int.?int.!int}.end$ and establishes the pattern to be followed by the server at its own endpoint of the channel. Duality guarantees the absence of *communication errors*.

If we assume these primitives are executed at different sites, then the locality principle is seen to fail. We introduce an asynchronous connection mechanism whereby the connection request on a is buffered at the local sites of all the parties participating (as described below) in the session s. A similar treatment is given to language expressions for sending/receiving values and selection/branching. Before providing further details on how asynchronous connection is established, we discuss what form session types take in DCMS.

Session types in DCMS are *multipoint*: a channel has one positive or *master* endpoint and one or more negative or *slave* endpoints. Each input/output or branch/select type (see Sec. 3) in the sequence that makes up a session type is decorated with a label (a site name or site name variable) indicating the referenced site. As an example, consider the system $cl[\![e_1]\!] \parallel atm[\![e_2]\!] \parallel bk[\![e_3]\!]$, adapted from [BCG05], where cl, atm and bk stand for client, ATM and bank, resp. Consider the following session type s for a:

$$
\begin{array}{l}
?^{cl}\mathsf{int}.\&^{cl}\{ \\
\quad \mathtt{withdraw} : ?^{cl}\mathsf{int}.!^{bk}\mathsf{int}. \oplus^{bk} \{\mathtt{withdraw} : !^{bk}\mathsf{int}.?^{bk}\mathsf{int}.!^{cl}\mathsf{int}.end, \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \diamond\,\mathtt{balance} : end\}, \\
\quad \diamond\,\mathtt{balance} : !^{bk}\mathsf{int}. \oplus^{bk} \{\mathtt{balance} : ?^{bk}\mathsf{int}.!^{cl}\mathsf{int}.end, \\
\qquad\qquad\qquad\qquad\qquad\quad \diamond\,\mathtt{withdraw} : end\} \\
\quad \}
\end{array}
$$

It reflects the pattern from the view of the *atm* and is the type assigned to the master endpoint (the types of the other endpoints are discussed below). The ATM first expects an integer from the client (an id) and then an indication as to whether a withdrawal or a balance request is required. In the case of the former (the latter is described similarly), the amount is expected from the client after which this amount is sent to the bank followed by an indication that the client has requested a withdrawal. Note that the type *end* for `balance` indicates that this branch is not available for selection *here*. Other occurrences of the `balance` branch may have a type different from *end*, however all different uses of this branch should be compatible: any two non-end types should be the same. This encoding of multiple uses of branches in multipoint session types allows a higher degree of expressiveness not readily available in standard (binary) session types without adding new features (cf.[BCG05]): indeed, although this example could be presented using binary session types, it is at a loss in precision (for example, after receiving a withdraw request from the client, the ATM could issue multiple withdrawl requests from the bank without violating the patterns described by the binary session types).

$$\left\langle \begin{array}{l} !^{bk}\mathsf{int}.\oplus^{bk}\{\mathtt{withdraw} : !^{bk}\mathsf{int}.?^{bk}\mathsf{int}.end, \\ \qquad\qquad \diamond\mathtt{balance} : end\} \end{array}, \begin{array}{l} !^{bk}\mathsf{int}.\oplus^{bk}\{\mathtt{balance} : ?^{bk}\mathsf{int}.end, \\ \qquad\qquad \diamond\mathtt{withdraw} : end\} \end{array} \right\rangle$$

$$\begin{array}{l} ?^{atm}\mathsf{int}.\&^{atm}\{\mathtt{withdraw} : ?^{atm}\mathsf{int}.!^{atm}\mathsf{int}.end, \\ \qquad\qquad\qquad \diamond\mathtt{balance} : !^{atm}\mathsf{int}.end\} \end{array}$$

Fig. 1. Compatible session types

Returning to the discussion on connecting through ports, recall that the request primitive buffers a request on port a at each of the participating sites of the session type s. Each of these sites may agree to participate by issuing a accept primitive on a with some session type s'. It should be mentioned that we do not require that all participating sites issue an accept before engaging in communication. This reduces the possibility of stuck systems due to the absence or reluctance of a participating site to engage.

Suppose n is one of these participating sites (in our example, apart from atm, they are cl and bk as may be read off from s). In order to guarantee the absence of communication errors, the *part of* s that pertains to n (called the restriction of s to n) should be compared for duality with s'. This requires that all uses of the same branch be compatible, as mentioned above. In our example the restriction of s to bk yields the set of session types in Fig. 1(top) which, if compatible, allows the desired restriction to be obtained (Fig. 1(bottom)). These concepts are precised below.

Finally, the main ingredient in the proof of Communication Safety (Prop. 3) and Subject Reduction (Prop. 1) is the notion of *duality invariant*. As execution progresses session types pending consumption together with the values already sent out and residing in buffers distributed over the system are synthesized into sequences of types and values which we dub *trace types*. Trace types are compared using a binary relation that takes into account the asynchronous nature of communication. Subject Reduction is then formulated as the property that this invariant is upheld during reduction.

Structure of the paper. Sec. 2 introduces the syntax of DCMS together with its operational semantics. Types and typing rules are presented in Sec. 3. Here we also discuss compatibility and duality. Sec. 4 introduces the duality invariant and addresses Subject Reduction and Safety. Finally, we conclude and offer avenues for further research.

2 Syntax and Operational Semantics

2.1 Syntax

The syntax of DCMS is presented in Fig. 2. A *site* $n[\![e]\!]$ has a name n which ranges over a set of site names $m, n, \ldots$ and a *thread expression* e which is said to run at n. A *system* is a set of sites. For simplicity we assume all sites in a system to have different site names. Expressions may be one of the following. An identifier x; a value v (described below); a let expression $\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$ with the usual interpretation; a connection, communication or branching expression. A *connection expression* can be of one of two kinds: $\mathsf{request}\ a(u : s)\{e\}$ or $\mathsf{accept}\ a(u : s, d)\{e\}$. The former requests asynchronously on port a that a new multipoint channel be established for communication following pattern s. The latter accepts such a request and replaces u with its corresponding endpoint and d with the name of the requesting site. A *communication expression* can be either $\mathsf{send}(u, \lambda, e)$ or $\mathsf{receive}(u, \lambda)$. The former sends the value resulting from e over u to location λ, whereas the latter reads from its local buffer u a value expected from λ. A branching expression can either be a select $\langle u, \lambda\rangle \rhd l\ \mathsf{in}\ \{e\}$ or branch $\langle u, \lambda\rangle \lhd \{l_1 : e_1 \diamond \ldots \diamond l_n : e_n\}$ (abbreviated $\langle u, \lambda\rangle \lhd_{i=1,n} \{l_i = e_i\}$). The former selects a branch by sending (asynchronously) a label over u to site λ. The latter reads a label from its local buffer u to see if λ has selected a branch. In the case that the buffer is empty, execution is blocked until a label is received.

A *value* is the result of a computation. It can be either true, false or null, the null expression. The additional run-time value (shaded in the figure) l is also possible. This value is not part of the user syntax but arises as a consequence of the definition of the operational semantics. Connection request values are discussed in Sec. 2.2. We write $\overline{v}$ (resp. $\overline{r}$) for a sequence of values (resp. connection request values) and ϵ for the empty such sequence. Also, $\overline{v}^R$ is the reverse sequence of $\overline{v}$.

We write $\textsc{fv}(e)$ for the free variables of e. In particular, let binds the declared variable; in both $\mathsf{request}$ and accept u is bound in e, in accept d is also bound in s and e. Also, $e_1; e_2$ is shorthand for $\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$ with $x \notin \textsc{fv}(e_2)$. We write $e\{x \mapsto v\}$ for the capture-free substitution of all free occurrences of x in e by v. Expressions are identified modulo renaming of bound variables.

2.2 Operational Semantics

The operational semantics of DCMS is described in terms of a *global buffer*. A global buffer (written h) associates a mapping describing the contents of its local port and local channel buffers to each location. We write h_n for the mapping for site n. A *port buffer* for a, denoted $h_n(a)$, is a sequence of connection request values $k^+@n$. The expression $k^+@n$ in the port buffer indicates the request by a foreign party n to establish a session of type s. A *channel buffer* for $k^p\mathbb{F}m$, denoted $h_n(k^p\mathbb{F}m)$, is a sequence of values received so far from location m via channel k^p.

Reduction schemes are presented in two groups: Fig. 3 presents those for expressions and Fig. 4 those for sites. A $\mathsf{request}$ expression adds a request to the buffer for port a at each of the sites participating in session type s. This set of sites is written $\textsc{Participants}(s)$ and simply collects the set of all site names occurring in s. Additionally, a new empty channel buffer is locally created for each of the participating parties in preparation for receiving values from them. Finally, note that k^+ is required to be *locally* fresh in the sense that it has not been used as the master endpoint of a previously established

System	$S ::= n[\![e]\!]$	site
	$\mid\ S \parallel S$	distributed sites
Thread Expression	$e ::= x \mid v \mid$ let $x = e$ in e	
	$\mid$ request $a(u : s)\{e\}$	
	$\mid$ accept $a(u : s, d)\{e\}$	
	$\mid$ send(u, λ, e)	
	$\mid$ receive(u, λ)	
	$\mid\ \langle u, \lambda\rangle \rhd l$ in $\{e\}$	
	$\mid\ \langle u, \lambda\rangle \lhd \{l_1 : e_1 \diamond \ldots \diamond l_n : e_n\}$	
Site Name	$\lambda ::= m \mid n \mid \ldots$	site name
	$\mid\ d$	site variable
Port	$a, b, \ldots$	
Polarity	$p ::= + \mid -$	
(Polarized) Channel	$u ::= k^p$	channel
	$\mid\ x \mid y \mid \ldots$	channel variable
Value	$v ::=$ true $\mid$ false $\mid$ null $\mid\ l$	
Conn. Request Value	$r ::= k^+@n$	
Heap	$h ::= [] \mid h \cdot [(m)(a) \mapsto \overline{r}]$	
	$\mid\ h \cdot [(m)(k^p\mathbb{F}n) \mapsto \overline{v}]$	

Fig. 2. Syntax

connection at that site. We write $h \cdot [(m_i)(a) \mapsto \overline{r}_i]_{i\in 1..o}$ as a shorthand for $h \cdot [(m_1)(a) \mapsto \overline{r}_1] \ldots [(m_o)(a) \mapsto \overline{r}_o]$. Likewise $h \cdot [(n)(k^p\mathbb{F}m_i) \mapsto \epsilon]_{i\in 1..o}$ stands for $h \cdot [(n)(k^p\mathbb{F}m_1) \mapsto \epsilon] \cdot \ldots \cdot [(n)(k^p\mathbb{F}m_o) \mapsto \epsilon]$.

The accept expression requires a pending connection request to be available at its local buffer for port a. It then creates a new local channel buffer for communication with the master endpoint and updates its local port buffer by removing the request. The k^- endpoint is assumed to be locally fresh for otherwise reduction blocks. The asynchronous send expression adds the value v to the local channel buffer of the corresponding endpoint. The result of executing a send expression is null. The receive expression blocks until a value is available at the appropriate local buffer and then reads it. The schemes for select and branch are similar to send and receive except that labels are sent or received rather than arbitrary values (and the appropriate branch is selected). Finally, there are two congruence schemes for reducing in the declaration part of a let expression and inside the last argument of a send expression.

Reduction schemes for sites are standard. Note that, as usual, reduction is modulo structural congruence ($\equiv$) rules.

3 Type System

Typing judgements for thread expressions and sites are $\Gamma; \Sigma \blacktriangleright_n e : t; \Sigma'$ and $\Gamma; \Sigma \blacktriangleright S : \Sigma'$, resp. The *standard environment* Γ maps standard types to

[Request-R]

$$\begin{array}{c}\mathsf{request}\ a(u:s)\{e\}, h \cdot [(m_i)(a) \mapsto \overline{r}_i]_{i\in 1..k}\\ \longrightarrow_n \\ e\{u \mapsto k^+\}, h \cdot [(m_i)(a) \mapsto k^+@n \cdot \overline{r}_i]_{i\in 1..k} \cdot [(n)(k^+\mathbb{F}m_i) \mapsto \epsilon]_{i \in 1..k}\end{array}$$

where $\textsc{Participants}(s) = \{m_1, \ldots, m_k\}$ and $k^+ \notin h_n$.

[Accept-R]

$$\begin{array}{c}\mathsf{accept}\ a(u:s,d)\{e\}, h \cdot [(n)(a) \mapsto \overline{r} \cdot k^+@m]\\ \longrightarrow_n \\ e\{u \mapsto k^-\}\{d \mapsto m\}, h \cdot [(n)(a) \mapsto \overline{r}] \cdot [(n)(k^-\mathbb{F}m) \mapsto \epsilon]\end{array}$$

where $k^- \notin h_n$.

[Send-R]

$$\begin{array}{c}\mathsf{send}(k^p, m, v), h \cdot [(m)(k^{\overline{p}}\mathbb{F}n) \mapsto \overline{v}]\\ \longrightarrow_n \\ \mathsf{null}, h \cdot [(m)(k^{\overline{p}}\mathbb{F}n) \mapsto v \cdot \overline{v}]\end{array}$$

[Rcv-R]

$$\begin{array}{c}\mathsf{receive}(k^p, m), h \cdot [(n)(k^p\mathbb{F}m) \mapsto \overline{v} \cdot v]\\ \longrightarrow_n \\ v, h \cdot [(n)(k^p\mathbb{F}m) \mapsto \overline{v}]\end{array}$$

[Select-r]

$$\begin{array}{c}k^p@m \rhd l_i\ \mathsf{in}\ \{e\}, h \cdot [(m)(k^{\overline{p}}\mathbb{F}n) \mapsto \overline{v}]\\ \longrightarrow_n \\ e, h \cdot [(m)(k^{\overline{p}}\mathbb{F}n) \mapsto l_i \cdot \overline{v}]\end{array}$$

[Branch-R]

$$\begin{array}{c}k^p@m \lhd \{\diamond_{i=1,n} l_i : e_i\}, h \cdot [(n)(k^p\mathbb{F}m) \mapsto \overline{v} \cdot l_i]\\ \longrightarrow_n \\ e_i, h \cdot [(n)(k^p\mathbb{F}m) \mapsto \overline{v}]\end{array}$$

[Let-R]

$$\begin{array}{c}\mathsf{let}\ x = v\ \mathsf{in}\ e, h\\ \longrightarrow_n \\ e\{x \mapsto v\}, h\end{array}$$

[CongLet-R]

$$\frac{e_1, h \longrightarrow_n e_1', h'}{\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2, h \longrightarrow_n \mathsf{let}\ x = e_1'\ \mathsf{in}\ e_2, h'}$$

[CongSend-R]

$$\frac{e, h \longrightarrow_n e', h'}{\mathsf{send}(k^p, m, e), h \longrightarrow_n \mathsf{send}(k^p, m, e'), h'}$$

Fig. 3. Expression Reduction Schemes

term variables and ports and the *session environment* Σ maps located channels (i.e. expressions of the form $k^p@m$) to session types. Fig. 5 defines types and environments. We assume $a \notin \textsc{Dom}(\Gamma)$, $x \notin \textsc{Dom}(\Gamma)$ and $u \notin \textsc{Dom}(\Sigma)$ (i.e. $u@m \notin \textsc{Dom}(\Sigma)$ for any m). We write $\Sigma\{u \mapsto k^p\}$ for substitution of channel variable u by a channel k^p in environment Σ. Likewise, $\Sigma\{d \mapsto m\}$ stands for substitution of site variable d by a site name m in environment Σ. These notions are standard and hence their definitions omitted.

The aforementioned judgements are defined in terms of *typing rules* (Fig. 7 and 6). We only describe the interesting ones. However, before doing so, we need to provide a precise meaning to the part of a session type that pertains to a specific site. As mentioned, this part can only be computed if different uses of branches are *compatible*. Thus we first make this notion precise (Def. 2).

$$\begin{array}{c} S_1 \parallel S_2 \equiv S_2 \parallel S_1 \\ S_1 \parallel (S_2 \parallel S_3) \equiv (S_1 \parallel S_2) \parallel S_3 \\ S_1 \equiv S_2 \Rightarrow S \parallel S_1 \equiv S \parallel S_2 \end{array} \qquad \frac{e, h \longrightarrow_n e', h'}{n[\![e]\!], h \longrightarrow n[\![e']\!], h'}\,[\text{SITE-R}]$$

$$\frac{S, h \longrightarrow S', h'}{S \parallel S_1, h \longrightarrow S' \parallel S_1, h'}\,[\text{PAR-R}] \qquad \frac{S_1' \equiv S_1 \quad S_1, h \longrightarrow S_2, h' \quad S_2 \equiv S_2'}{S_1', h \longrightarrow S_2', h'}\,[\text{STR-R}]$$

Fig. 4. Site Reduction Schemes

Direction	$\dagger ::= ? \mid !$		
Partial Session Type	$\pi ::= \epsilon \mid \dagger^\lambda\, t.\pi$		
	$\mid\ \&^\lambda\{l_1 : s, \ldots, l_n : s\}$	Standard Env.	$\Gamma ::= \epsilon \mid \Gamma, a : s \mid \Gamma, x : t$
	$\mid\ \oplus^\lambda\{l_1 : s, \ldots, l_n : s\}$	Channel Env.	$\Sigma ::= \epsilon \mid \Sigma, u@n : s$
Session Type	$s ::= \pi.end$		
Standard Type	$t ::=$ bool $\mid$ cmd $\mid s$		

Fig. 5. Types

Definition 1 (Compatible Set). *A set of session types $\{s_1, \ldots, s_n\}$ is compatible if $s_1 \sqcup s_2 \sqcup \ldots \sqcup s_n$ is defined, where $\sqcup$ is the following commutative, associative operation:*

$$\begin{aligned} s \sqcup s &= s \\ end \sqcup s &= s \\ s \sqcup end &= s \\ \pi.end \sqcup \pi'.end &= (\pi \sqcup \pi').end \end{aligned}$$

$$\begin{aligned} \pi \sqcup \pi &= \pi \\ ?^\lambda t.\pi_1 \sqcup ?^\lambda t.\pi_2 &= ?^\lambda t.(\pi_1 \sqcup \pi_2) \\ !^\lambda t.\pi_1 \sqcup !^\lambda t.\pi_2 &= !^\lambda t.(\pi_1 \sqcup \pi_2) \\ \&^\lambda_{i=1,n}\{l_i : s_i\} \sqcup \&^\lambda_{i=1,n}\{l_i : s_i'\} &= \&^\lambda_{i=1,n}\{l_i : s_i \sqcup s_i'\} \\ \oplus^\lambda_{i=1,n}\{l_i : s_i\} \sqcup \oplus^\lambda_{i=1,n}\{l_i : s_i'\} &= \oplus^\lambda_{i=1,n}\{l_i : s_i \sqcup s_i'\} \end{aligned}$$

A session type is compatible when it is compatible from the viewpoint of all participating sites.

Definition 2 (Compatible session type). *A session type s is compatible if for all $m \in$ PARTICIPANTS(s), SIMPLIFY$(s \downarrow m)$ is compatible, where*

Typing Rules for Values

$$\frac{}{\Gamma; \Sigma \blacktriangleright_n \mathsf{null} : \mathsf{cmd}; \Sigma}\,[\textsc{null}] \qquad \frac{e \in \{\mathsf{true}, \mathsf{false}\}}{\Gamma; \Sigma \blacktriangleright_n e : \mathsf{bool}; \Sigma}\,[\textsc{True/False}]$$

Typing Rules for Sites

$$\frac{\Gamma; \Sigma \blacktriangleright_n e : t; \Sigma'}{\Gamma; \Sigma \blacktriangleright n[\![e]\!] : \Sigma'}\,[\textsc{StartSite}] \qquad \frac{\Gamma; \Sigma \blacktriangleright S_1 : \Sigma' \quad \Gamma; \Sigma' \blacktriangleright S_2 : \Sigma''}{\Gamma; \Sigma \blacktriangleright S_1 \parallel S_2 : \Sigma''}\,[\textsc{ParSite}]$$

Fig. 6. Typing Rules for Values and Sites

$$\begin{aligned}
(\pi.end) \downarrow m &= (\pi \downarrow m).end \\
(\dagger^\lambda t.\pi) \downarrow m &= \begin{cases} \dagger^\lambda t.(\pi \downarrow m) & \textit{if } \lambda = m \\ \pi \downarrow m & \textit{otherwise} \end{cases} \\
\&^\lambda_{i=1,n}\{l_i : s_i\} \downarrow m &= \begin{cases} \&^\lambda_{i=1,n}\{l_i : s_i \downarrow m\} & \textit{if } \lambda = m \\ \{s_1 \downarrow m, \dots, s_n \downarrow m\} & \textit{otherwise} \end{cases} \\
\oplus^\lambda_{i=1,n}\{l_i : s_i\} \downarrow m &= \begin{cases} \oplus^\lambda_{i=1,n}\{l_i : s_i \downarrow m\} & \textit{if } \lambda = m \\ \{s_1 \downarrow m, \dots, s_n \downarrow m\} & \textit{otherwise} \end{cases}
\end{aligned}$$

where SIMPLIFY(_) *rewrites its argument, in all subterms, using the following term rewrite rule until a normal form is reached*[1]*.*

$$\dagger^\lambda t.\{s_1, \dots, s_n\} \longrightarrow \{\dagger^\lambda t.s_1, \dots, \dagger^\lambda t.s_n\}$$

Under the assumption of compatibility we can define the *restriction* of a session type s to a site name m, for $m \in \textsc{Participants}(s)$, as $\sqcup(\textsc{Simplify}(s \downarrow m))$ and write $s \upharpoonright m$. Finally, we introduce the notion of dual session types, used to type the accept expression. It is the standard notion that may be found in the extant literature on the subject: session types s and s' are *dual* (or (m, n)-*dual* to be more precise) if the predicate $\textsc{Dual}(s, s')$ holds:

$$\begin{aligned}
&\textsc{Dual}(\epsilon, \epsilon) \text{ holds} \\
\textsc{Dual}(\pi.end, \pi'.end) &= \textsc{Dual}(\pi, \pi') \\
\textsc{Dual}(?^n t.\pi, !^m t.\pi') &= \textsc{Dual}(\pi, \pi') \\
\textsc{Dual}(!^n t.\pi, ?^m t.\pi') &= \textsc{Dual}(\pi, \pi') \\
\textsc{Dual}(\&^n_{i=1,p}\{l_i : s_i\}, \oplus^m_{i=1,p}\{l_i : s'_i\}) &= \textstyle\bigwedge_{i=1,p} \textsc{Dual}(s_i, s'_i) \\
\textsc{Dual}(\oplus^n_{i=1,p}\{l_i : s_i\}, \&^m_{i=1,p}\{l_i : s_i\}) &= \textstyle\bigwedge_{i=1,p} \textsc{Dual}(s_i, s'_i)
\end{aligned}$$

The typing rules for values are standard, as are those for variables and let expressions. Note that the session environment remains unmodified in the case of values and variables given that these expressions themselves do not perform operations involving channels. A request on port a requires the type of a to be

[1] Uniqueness of normal forms follows from orthogonality and termination.

$$\frac{}{\Gamma, x : t; \Sigma \blacktriangleright_n x : t; \Sigma}\,[\textsc{Var}] \qquad \frac{\Gamma; \Sigma \blacktriangleright_n e : t; \Sigma' \quad \Gamma, x : t; \Sigma' \blacktriangleright_n e' : t'; \Sigma''}{\Gamma; \Sigma \blacktriangleright_n \mathsf{let}\ x = e\ \ \mathsf{in}\ e' : t'; \Sigma''}\,[\textsc{Let}]$$

$$\frac{\Gamma, a : s; \Sigma, u@n : s \blacktriangleright_n e : t; \Sigma', u@n : end}{\Gamma, a : s; \Sigma \blacktriangleright_n \mathsf{request}\ a(u : s)\{e\} : t; \Sigma'}\,[\textsc{Request}]$$

$$\frac{\begin{array}{c}\textsc{Dual}(s \restriction n, s'\{d \mapsto m\}) \\ m \text{ fresh} \\ \Gamma, a : s; \Sigma, u@n : s' \blacktriangleright e : t; \Sigma', u@n : end\end{array}}{\Gamma, a : s; \Sigma \blacktriangleright_n \mathsf{accept}\ a(u : s', d)\{e\} : t; \Sigma'}\,[\textsc{Accept}]$$

$$\frac{\Gamma; \Sigma \blacktriangleright_n e : t; \Sigma', u@n : !^{\lambda}t.s}{\Gamma; \Sigma \blacktriangleright_n \mathsf{send}(u, \lambda, e) : \mathsf{cmd}; \Sigma', u@n : s}\,[\textsc{Send}]$$

$$\frac{}{\Gamma; \Sigma, u@n : ?^{\lambda}t.s \blacktriangleright_n \mathsf{receive}(u, \lambda) : t; \Sigma, u@n : s}\,[\textsc{Receive}]$$

$$\frac{\Gamma; \Sigma, u@n : s_i \blacktriangleright_n e : t; \Sigma'}{\Gamma; \Sigma, u@n : \oplus^{\lambda}_{i=1,n}\{l_i : s_i\}\} \blacktriangleright_n \langle u, \lambda \rangle \rhd l\ \mathsf{in}\ \{e\} : t; \Sigma'}\,[\textsc{Select}]$$

$$\frac{\Gamma; \Sigma, u@n : s_i \blacktriangleright_n e_i : t; \Sigma'}{\Gamma; \Sigma, u@n : \&^{\lambda}_{i=1,n}\{l_i : s_i\} \blacktriangleright_n \langle u, \lambda \rangle \lhd \{\diamond_{i=1,n} l_i = e_i\} : t; \Sigma'}\,[\textsc{Branch}]$$

Fig. 7. Typing Rules for Expressions

declared globally with some session type s. The session environment is augmented with a new located channel (i.e. expression of the form $u@n$) before typing the body e. The type of the request expression is that of its body. Finally, the located channel is assumed to be completely consumed within this body. A accept also augments the session environment before typing its body, however it uses the declared type s'. A check is performed to verify whether the session type of a restricted to n, the site hosting the accept expression, is dual to s' (prior application of the substitution $\{d \mapsto m\}$). Given that the name of the site requesting the request is unknown, a fresh name is substituted for all occurrences of the site name variable d in s'. A $\mathsf{send}(u, \lambda, e)$ expression requires that we first type e. The resulting session environment should include a session type for $u@n$ with a output type expression at the head. The type of e and the one declared in the output type should agree. Also, the label of the output type should agree with the destination declared in the send expression. The remaining typing rules may be understood along similar lines.

We conclude this section with a standard property of type systems also shared by DCMS.

Lemma 1 (Substitution Preserves Typing)

1. *$\Gamma; \Sigma_1 \blacktriangleright_n e : t; \Sigma_2$ and $k^p \notin \Sigma_1$ implies $\Gamma; \Sigma_1\{u \mapsto k^p\} \blacktriangleright_n e\{u \mapsto k^p\} : t; \Sigma_2\{u \mapsto k^p\}$.*
2. *$\Gamma; \Sigma_1 \blacktriangleright_n e : t; \Sigma_2$ and $m \neq n$ implies $\Gamma; \Sigma_1\{d \mapsto m\} \blacktriangleright_n e\{d \mapsto m\} : t; \Sigma_2\{d \mapsto m\}$.*

4 Subject Reduction and Safety

This section addresses Subject Reduction (SR) and Safety. The latter states that the type system guarantees the absence of communication errors while the former ensures that reduction preserves this state of affairs. We consider a communication error to be an execution state where a site attempts to read a value from its local buffer with the wrong type. In order to prove the absence of such errors, we must take into consideration how the system evolves during computation. During the course of reduction, values are sent out to local buffers distributed throughout the system. Accordingly, the types of channels are consumed. Therefore, both session types and the contents of buffers must be taken into account in order to determine safety. *Trace types* merge session types and values and are defined by the grammar on the left:

$$\begin{array}{rll} \tau ::= & end & \\ | & \epsilon & \qquad O ::= \Box_{n\in\mathbb{N}} \\ | & \dagger t.\tau & \qquad\quad | \ !t.O \\ | & v.\tau & \qquad\quad | \ v.O \\ | & \&_{i=1,n}\{l_i : \tau_i\} & \qquad\quad | \ \oplus_{i=1,n}\{l_i : O_i\} \\ | & \oplus_{i=1,n}\{l_i : \tau_i\} & \end{array}$$

The absence of site names in $\dagger$, & and $\oplus$ allows for a conciser presentation (correspondence between site names is guaranteed by the typing rule for accept). The grammar on the right defines *trace-output contexts*. In an asynchronous setting sending a value is a non-blocking operation and hence trace-output contexts represent the activity that could take place before a blocking operation is executed. An example trace-output context is $O = 3. \oplus \{l_1 : \Box_1, l_2 : l_j.\Box_2, l_3 : \Box_3\}$ (assuming we may send integers): a 3 may be sent followed by one of l_1, l_2, l_3, followed by l_j in the case that l_2 was selected. Note that output-trace contexts may have more than one occurrence of a hole. Holes are indexed with a unique index indicated with a natural number as subscript. We write $O[\tau_1, \ldots, \tau_n]$ or simply $O[\boldsymbol{\tau}]_{k=1,n}$ for the result of filling in holes $\Box_1$ to $\Box_n$ with τ_1 to τ_n, resp. We often omit the subscript in $O[\boldsymbol{\tau}]_{k=1,n}$ (and write $O[\boldsymbol{\tau}]$) for the sake of readability.

Both trace types and trace-output contexts are used for stating the duality invariant, as motivated above. A further word on notation: $v : t$ is a shorthand for $\emptyset; \emptyset \blacktriangleright_n v : t; \emptyset$, for any n.

Definition 3 (A-Duality of trace types). *The binary relation on trace types called a(synchronous)-duality is defined inductively as follows:*

$$\frac{}{end \bowtie end}\,[\text{END/END-D}] \qquad \frac{}{\epsilon \bowtie \epsilon}\,[\epsilon/\epsilon\text{-D}]$$

$$\frac{\sigma \bowtie \tau \quad v : t}{?t.\sigma \bowtie v.\tau}\,[?/v\text{-D}] \qquad \frac{\sigma \bowtie \tau}{?t.\sigma \bowtie !t.\tau}\,[?/!\text{-D}]$$

$$\frac{\sigma \bowtie O[\boldsymbol{\tau}] \quad v : t}{v.\sigma \bowtie O[?t.\boldsymbol{\tau}]}\,[v/?\text{-D}] \qquad \frac{\sigma \bowtie O[\boldsymbol{\tau}]}{!t.\sigma \bowtie O[?t.\boldsymbol{\tau}]}\,[!/?\text{-D}]$$

$$\frac{\sigma_j \bowtie \tau \quad j \in 1..n}{\&_{i=1,n}\{l_i : \sigma_i\} \bowtie l_j.\tau}\,[\&/l\text{-D}] \quad \frac{\sigma_i \bowtie \tau_i \quad \textit{for each } i \in 1..n}{\&_{i=1,n}\{l_i : \sigma_i\} \bowtie \oplus_{i=1,n}\{l_i : \tau_i\}}\,[\&/\oplus\text{-D}]$$

$$\frac{\sigma \bowtie O[\boldsymbol{\tau_j}]_{k=1,o} \quad j \in 1..n_k}{l_j.\sigma \bowtie O[\boldsymbol{\&_{i=1,n}\{l_i : \tau_i\}}]_{k=1,o}}\,[l/\&\text{-D}]$$

$$\frac{\sigma_i \bowtie O[\boldsymbol{\tau_i}]_{k=1,o} \quad \textit{for each } i \in 1..n}{\oplus_{i=1,n}\{l_i : \sigma_i\} \bowtie O[\&_{i=1,n}\{l_i : \boldsymbol{\tau_i}\}]_{k=1,o}}\,[\oplus/\&\text{-D}]$$

If $\sigma \bowtie \tau$, then we say σ is *a-dual* to τ. The intuition behind $\sigma \bowtie \tau$ is that σ is the session type of one endpoint of a session *including* the values this endpoint already sent out and likewise for the other endpoint τ. If they are both *end* or ϵ, then they are said to agree. If σ expects to receive a value of type t, then either it has already been sent ([?/v-D]) or the send operation is next in line according to the session type of τ ([?/!-D]). If σ has sent out a value ([v/?-D]), then τ must be prepared to read but not necessarily immediately. Indeed, first it may send out some other values (represented by the trace-output context O). Note that $O[?t.\boldsymbol{\tau}]_{k=1,o}$ in [v/?-D] and [!/?-D] means $O[?t.\tau_1, \ldots, ?t.\tau_o]$. The remaining rules follow similar arguments.

Let $|s|$ stand for the trace type resulting from erasing all site name information from session type s. Note that dual session types are a-dual, as may be verified by induction on s:

Lemma 2. *Let s, s' be session types. Then* $\textsc{Dual}(s, s')$ *implies* $|s| \bowtie |s'|$.

There are, of course, a-dual session types that are not dual. For example, we have $!t.?t'.end \bowtie !t'.?t.end$, for any t, t', however for no decoration of site names shall these types become dual. A-duality shares another property of duality, namely symmetry.

Lemma 3 (Symmetry of $\bowtie$)

1. $O[\boldsymbol{\sigma}]_{k=1,o} \bowtie \tau$ *implies*
   (a) $O[?t.\boldsymbol{\sigma}]_{k=1,o} \bowtie v.\tau$, *if* $v : t$.
   (b) $O[?t.\boldsymbol{\sigma}]_{k=1,o} \bowtie !t.\tau$,
   (c) $O[\boldsymbol{\&_{i=1,n}\{l_i : \rho_i\}}]_{k=1,o} \bowtie l_j.\tau$, *where* $\rho_{j_k} = \sigma_k$ *for each* $k \in 1..o$, *if* $j \in 1..n$.
2. $\sigma \bowtie \tau$ *implies* $\tau \bowtie \sigma$.

Proof. The first item is by induction on the structure of O and the second by induction on the derivation of $\sigma \bowtie \tau$ and resorts to the first one.

Some further properties of $\bowtie$ shall be useful. Items (1) and (2) below are used in the proof of Subject Reduction to show that the duality invariant (stated below) is upheld after a send and receive expression has been executed. Items (3) and (4) are required for the case of select and branch. The proof of all items is by induction on the length of $\overline{v}$.

Lemma 4. *1. $\overline{v}.!t.\tau_1 \bowtie \tau_2$ and $v : t$ imply $\overline{v}.v.\tau_1 \bowtie \tau_2$.*
2. $\overline{v}.?t.\tau_1 \bowtie v.\tau_2$ implies $v : t$ and $\overline{v}.\tau_1 \bowtie \tau_2$.
3. $\overline{v}. \oplus_{i=1,n} \{l_1 : \tau_i\} \bowtie \sigma$ and $j \in 1..n$ imply $\overline{v}.l_j.\tau_j \bowtie \sigma$.
4. $\overline{v}.\&_{i=1,n}\{l_i : \tau_i\} \bowtie l_j.\sigma$ implies $j \in 1..n$ and $\overline{v}.\tau_j \bowtie \sigma$.

Let us illustrate the first item with a concrete example. Consider the reduction of the expression $\mathsf{send}(k^p, m, v)$ at site n. The trace type $\overline{v}.!t.\tau_1 \bowtie \tau_2$ will be interpreted as the view of k^p at n as follows:

– $\overline{v}$ is the sequence of values already sent out by n on k^p to m and not consumed, and
– $!t.\tau_1$ is the channel type of k^p at n (with its site names erased).

Assuming this view is a-dual to that of m (represented by τ_2), Lemma 4(1) states that replacing $!t$ by v (the value sent by n on k^p) preserves a-duality.

Definition 4 (Duality Invariant). *A pair of session environment and global buffer satisfy the duality invariant, written* $\textsc{DualityInv}(\Sigma; h)$, *if* $k^+@n : s_n \in \Sigma$ *and* $k^-@m : s_m \in \Sigma$ *implies*

$$\overline{v}^R.|s_n \upharpoonright m| \bowtie \overline{w}^R.|s_m|$$

where $h_n(k^+\mathbb{F}m) = \overline{w}$ *and* $h_m(k^-\mathbb{F}n) = \overline{v}$.

One final ingredient is required before formulating our main result. Given that $\mathsf{request}$ and accept expressions create new communication channels, session environments may grow as reduction proceeds. Therefore, we define $\Sigma \leq \Sigma'$ as the smallest partial order that contains $\Sigma, u@\lambda : end \geq \Sigma$. The following property relating this partial order and typability is seen to hold.

Lemma 5 (Weakening)

1. $\Gamma; \Sigma_1 \blacktriangleright_n e : t; \Sigma_2$ and $\Sigma_1' \geq \Sigma_1$ imply $\Gamma; \Sigma_1' \blacktriangleright_n e : t; \Sigma_2'$, for some $\Sigma_2' \geq \Sigma_2$.
2. $\Gamma; \Sigma_1 \blacktriangleright S : \Sigma_2$ and $\Sigma_1' \geq \Sigma_1$ imply $\Gamma; \Sigma_1' \blacktriangleright S : \Sigma_2'$, for some $\Sigma_2' \geq \Sigma_2$.

Proposition 1 (SR for Expressions). $\Gamma; \Sigma_1 \blacktriangleright_n e : t; \Sigma_2$ *and* $\textsc{DualityInv}(\Sigma_1; h)$ *and* $e, h \longrightarrow_n e', h'$ *implies* $\Gamma; \Sigma_1' \blacktriangleright_n e' : t; \Sigma_2'$ *and* $\textsc{DualityInv}(\Sigma_1'; h')$, *for some* Σ_1' *and* $\Sigma_2' \geq \Sigma_2$.

Proof. By induction on the derivation of $e, h \longrightarrow_n e', h'$. We include a sample case, namely that of a [SEND-R] reduction step.

– $e = \mathsf{send}(k^p, m, v)$ and $h = h'' \cdot [(m)(k^{\overline{p}}\mathbb{F}n) \mapsto \overline{v}]$,
– $e' = \mathsf{null}$ and $h' = h'' \cdot [(m)(k^{\overline{p}}\mathbb{F}n) \mapsto v \cdot \overline{v}]$.

From $\Gamma; \Sigma_1 \blacktriangleright_n \mathsf{send}(k^p, m, v) : \mathsf{cmd}; \Sigma_2$ we deduce

1. $\Sigma_1 = \Sigma_{11}, k^p@n : !^m t.s$ and
2. $\Sigma_2 = \Sigma_{11}, k^p@n : s$.

Set $\Sigma_1' = \Sigma_{11}, k^p@n : s$ and $\Sigma_2' = \Sigma_2 (= \Sigma_1')$. Then note that

3. $\Gamma; \Sigma_{11}, k^p@n : s \blacktriangleright_n \mathsf{null} : \mathsf{cmd}; \Sigma_{11}, k^p@n : s$ is immediate and also
4. $\textsc{DualityInv}(\Sigma_{11}, k^p@n : s; h')$.

We develop (4). Suppose $p = +$ and $k^-@m : s_m \in \Sigma_1'$. Then also $k^-@m : s_m \in \Sigma_1$ and from $\textsc{DualityInv}(\Sigma_1; h)$:

$$\overline{v}^R.|!^m t.s \upharpoonright m| \bowtie \overline{w}^R.|s_m|$$

where $h_n(k^+\mathbb{F}m) = \overline{w}$. By Lemma 4(1),

$$\overline{v}^R.v.|s \upharpoonright m| \bowtie \overline{w}^R.|s_m|$$

Suppose now that $p = -$ and $k^+@m : s_m \in \Sigma_1'$. Then also $k^+@m : s_m \in \Sigma_1$ and from $\textsc{DualityInv}(\Sigma_1; h)$:

$$\overline{w}^R.|s_m \upharpoonright n| \bowtie \overline{v}^R.|!^m t.s|$$

where $h_n(k^-\mathbb{F}m) = \overline{w}$. We resort to symmetry of $\bowtie$ (Lemma 3), followed by Lemma 4(1), and finally symmetry again.

Prop. 1 holds for sites too. This requires first showing that:

Lemma 6 (Structural Congruence Preserves Typability). *$\Gamma; \Sigma_1 \blacktriangleright S : \Sigma_2$ and $S \equiv S'$ implies $\Gamma; \Sigma_1 \blacktriangleright S' : \Sigma_2$.*

We can then obtain the desired extension.

Proposition 2 (SR for Sites). *$\Gamma; \Sigma_1 \blacktriangleright S : \Sigma_2$ and* $\textsc{DualityInv}(\Sigma_1; h)$ *and $S, h \longrightarrow S', h'$ implies $\Gamma; \Sigma_1' \blacktriangleright S' : \Sigma_2'$ and* $\textsc{DualityInv}(\Sigma_1'; h')$*, for some Σ_1' and $\Sigma_2' \geq \Sigma_2$.*

Proof. By induction on the derivation of $S, h \longrightarrow S', h'$.

– [Site-R]. Then $S = n[\![e]\!]$, $S' = n[\![e']\!]$ and $e, h \longrightarrow_n e', h'$. Also, $\Gamma; \Sigma_1 \blacktriangleright_n e : t; \Sigma_2$ for some t. We conclude by resorting to Subject Reduction for Expressions.
– [Par-R]. Then $S = S_1 \parallel S_2$, $S' = S_1' \parallel S_2$ and $S_1, h \longrightarrow S_1', h'$. Also, there exists Σ_3 such that $\Gamma; \Sigma_1 \blacktriangleright S_1 : \Sigma_3$ and $\Gamma; \Sigma_3 \blacktriangleright S_2 : \Sigma_2$. By the IH there exists Σ_1', Σ_3' such that $\Sigma_3' \geq \Sigma_3$ and $\Gamma; \Sigma_1' \blacktriangleright S_1 : \Sigma_3'$ and $\textsc{DualityInv}(\Sigma_1'; h')$. We conclude by Lemma 5.
– [Str-R]. Then there exist S_1, S_2 such that
  1. $S \equiv S_1$,

2. $S_1, h \longrightarrow S_2, h'$ and
3. $S_2 \equiv S'$.

From Lemma 6, $\Gamma; \Sigma_1 \blacktriangleright S_1 : \Sigma_2$. By IH, there exist Σ_1' and Σ_2' such that

4. $\Sigma_2' \geq \Sigma_2$,
5. $\Gamma; \Sigma_1' \blacktriangleright S_2 : \Sigma_2'$ and
6. $\textsc{DualityInv}(\Sigma_1'; h')$.

We conclude from (3) and Lemma 6.

In order to formally state Communication Safety we first introduce the convenient notion of evaluation context E:

$$E ::= \Box \mid \mathsf{let}\ x = E \ \ \mathsf{in}\ e \mid \mathsf{send}(k^p, m, E)$$

The hole in an evaluation environment singles out the part of the context where the redex involved in the next reduction step is located. Communication Safety says that if receive is the next expression to be reduced at some site n, then either the value expected has not been sent by the expected party yet and the channel type of this party coincides with the one expected by the receive, or the value is located in n's local buffer and has the expected type. Similarly for a branch expression.

Proposition 3 (Communication Safety). *Suppose $\Gamma; \Sigma_1 \blacktriangleright_n e : t; \Sigma_2$ and $\textsc{DualityInv}(\Sigma_1; h)$.*

1. *If $e = E[\mathsf{receive}(k^p, m)]$ and $k^{\overline{p}}@m \in \Sigma_1$, then $\Sigma_1(k^p@n) = ?^m t.s_n$ and $\Sigma_1(k^{\overline{p}}@n) = s_m$, for some session types s_n, s_m, and*

$$\overline{v}^R.|?^m t.s_n \restriction m| \bowtie \overline{w}^R.|s_m|$$

where $h_n(k^p \mathbb{F} m) = \overline{w}$ and $h_m(k^{\overline{p}} \mathbb{F} n) = \overline{v}$, and one of two cases holds
   - *(a) either $\overline{w}^R = \epsilon$ and $s_m = !^n t.s_m'$, for some s_m',*
   - *(b) or $\overline{w}^R = w.\overline{w}'$ and $w : t$, for some w and $\overline{w}'$.*
2. *If $e = E[\langle k^p, m\rangle \lhd_{i=1,o} \{l_i = e_i\}]$ and $k^{\overline{p}}@m \in \Sigma_1$, then $\Sigma_1(k^p@n) = \&^m_{i=1,o}\{l_i : s_i'\}$ and $\Sigma_1(k^{\overline{p}}@n) = s_m$, for some session types s_i', s_m, and*

$$\overline{v}^R.|\&^m_{i=1,o}\{l_i : s_i'\} \restriction m| \bowtie \overline{w}^R.|s_m|$$

where $h_n(k^p \mathbb{F} m) = \overline{w}$ and $h_m(k^{\overline{p}} \mathbb{F} n) = \overline{v}$, and one of two cases holds
   - *(a) either $\overline{w}^R = \epsilon$ and $s_m = \oplus^n_{i=1,o}\{l_i : s_i'\}$, for s_i' with $i \in 1..o$,*
   - *(b) or $\overline{w}^R = l_j.\overline{w}'$ and $j \in 1..o$, for some $\overline{w}'$.*

The proof is by induction on E and relies on the following lemma:

Lemma 7. *1. $\overline{v}.?t.\sigma_1 \bowtie \overline{w}.\sigma_2$ implies*
   - *(a) either $\overline{v} = \epsilon$ and $\sigma_2 = !t.\sigma_2'$, for some σ_2',*
   - *(b) or $\overline{w} = w.\overline{w}'$ and $w : t$, for some $\overline{w}'$.*

2. $\overline{v}.\&_{i=1,o}\{l_i : s_i\} \bowtie \overline{w}.\sigma_2$ implies
   - *(a) either $\overline{w} = \epsilon$ and $\sigma_2 = \oplus_{i=1,o}\{l_i : s_i'\}$, for s_i' with $i \in 1..o$,*
   - *(b) or $\overline{w} = l_j.\overline{w}'$ and $j \in 1..o$, for some $\overline{w}'$.*

5 Related Work

Session types were introduced in work of Honda et al [Hon93, HKT94, HVK98]. Since then it has been studied in various programming language paradigms: π-calculus like [GH99, HG03, GVR03, BCG05, BCG04], mobile ambients [GCDC06], CORBA [VVR03], functional threads [VRG04] and for object-oriented programming [DCYAD05, DCMYD06]. Recent work [YV06] revisits Subject Reduction for session types in view of some subtle issues related with naming.

Dezani-Ciancaglini et al [DCYAD05] present a distributed object-oriented language with session types. Although they also deal with a system of named sites, they use synchronous communication. In later work [DCMYD06] they considered higher-order sessions for roughly the same language and study a progress property. Also, they introduce buffers to model the operational semantics. However, connection is still synchronous and no notion of multipoint session types is studied. The work of Neubauer and Thiemann [NT04] seems to be the first work on session types for asynchronous communication. They consider a functional programming language which, although lacks a notion of multipoint session type nor is distributed, introduces an interesting relation on values similar to a-duality. Session types for asynchronous communication in the setting of operating system services [FAH$^+$06] and object-oriented languages [CDCY07] has also been studied.

In recent [Yos07], independent work Honda, Yoshida and Carbone [HYC08] have developed a similar calculus of *multiparty* asynchronous session types. Interaction between participants is described by means of a "global type", essentially sequences of expressions of the form $p \rightarrow p' : k < U >$ expressing that "participant p sends a message of type U to channel k received by participant p'" (constructs for branching/selection and recursive types are also considered). Thus participants may share any number of channels, in contrast to our more restricted setting where only the master endpoint of a multipoint session type is shared. Since sharing gives rise to conflicts, a causality condition (dubbed "linearity" of global types) is required to ensure that global types are conflict-free. The remaining development is close to the one presented here: our notion of compatible session types corresponds to "coherence" of global types (Def. 4.2. in op. cit.), our duality invariant corresponds to "rollback of a message" (Sec. 5.2. in op. cit.). It should also be mentioned that Honda et al consider, in addition to Communication Safety, a progress property [DCMYD06]: roughly that, under certain conditions, a well-typed process that is ready to communicate shall always do so (Sec. 5.6. in op. cit.). We feel such a property should also hold for DCMS, although the details should be worked out.

6 Conclusions

We have presented a theory of session types for a core distributed calculus called DCMS. Distributed systems are represented as sets of named sites running threads. These sites communicate with each other by either sending/receiving values or selecting/branching on alternative code branches. The type system is built on the notion of session type: sequences of types of the entities being sent or received. The resulting session types are multipoint in the sense that they encode the interaction protocol to be followed by two or more parties. One such party is selected as a master and is the one that initiates a connection; multiple other parties are designated as slaves and each follow their own interaction scheme with the master. All communication expressions in DCMS are asynchronous: its semantics is described in terms of connection and communication buffers local to each site. Correctness of DCMS is proved in the form of a subject reduction theorem. This result consists in showing that a predicate on all buffers and the type assigned to each open connection called *duality invariant* is upheld at all times. This invariant roughly synthesizes run-time types, consisting of sequences of standard types and values, for channels and checks that any two endpoints have asynchronous dual such types. Asynchronous dual types is an extended notion of dual types that takes asynchronicity into account.

In order to bring out the fundamentals of combining session types and distributedness we have reduced our calculus to a minimal core. In particular, we have not included features such as run-time session type creation, sending/receiving session types, delegation of channels or spawning of new threads. This is left to future work. Type checking and inference based on the more lax notion of a-duality and an appropriate notion of subtyping should also be interesting avenues for further work.

Acknowledgements. To Healfdene Goguen and the referees for comments.

References

[BCG04] Bonelli, E., Compagnoni, A., Gunter, E.: Typechecking safe process synchronization. In: FGUC 2004, ENTCS. Elsevier, Amsterdam (2004)

[BCG05] Bonelli, E., Compagnoni, A., Gunter, E.: Correspondence assertions for process synchronization in concurrent communications. Journal of Functional Programming, Special issue on Language-Based Security, 15(2) (March 2005)

[Bou03] Boudol, G.: Mobile calculi based on domains: Core programming model v1. Technical Report Deliverable 1.2.1, MIKADO Global Computing Project IST-2001-32222 (2003)

[CDCY07] Coppo, M., Dezani-Ciancaglini, M., Yoshida, N.: Asynchronous session types and progress for object-oriented languages. In: Bonsangue, M.M., Johnsen, E.B. (eds.) FMOODS 2007. LNCS, vol. 4468, pp. 1–31. Springer, Heidelberg (2007)

[DCMYD06] Dezani-Ciancaglini, M., Mostrous, D., Yoshida, N., Drossopoulou, S.: Session types for object-oriented languages. In: Thomas, D. (ed.) ECOOP 2006. LNCS, vol. 4067, pp. 328–352. Springer, Heidelberg (2006)

[DCYAD05] Dezani-Ciancaglini, M., Yoshida, N., Ahern, A., Drossopoulou, S.: A distributed object-oriented language with session types. In: De Nicola, R., Sangiorgi, D. (eds.) TGC 2005. LNCS, vol. 3705, pp. 299–318. Springer, Heidelberg (2005)

[FAH$^+$06] Fähndrich, M., Aiken, M., Hawblitzel, C., Hodson, O., Hunt, G., Larus, J., Levi, S.: Language support for fast and reliable message-based communication in singularity os. In: Zwaenepoel, W. (ed.) EuroSys 2006. ACM SIGOPS, pp. 177–190. ACM Press, New York (2006)

[GCDC06] Garralda, P., Compagnoni, A., Dezani-Ciancaglini, M.: BASS: Boxed ambients with safe sessions. In: Maher, M. (ed.) PPDP 2006, pp. 61–72. ACM Press, New York (2006)

[GH99] Gay, S., Hole, M.: Types and subtypes for client-server interactions. In: Swierstra, S.D. (ed.) ESOP 1999 and ETAPS 1999. LNCS, vol. 1576, pp. 74–90. Springer, Heidelberg (1999)

[GVR03] Gay, S., Vasconcelos, V., Ravara, A.: Session types for inter-process communication. Technical Report TR-2003-133, Department of Computing Science, University of Glasgow (2003)

[HG03] Hole, M., Gay, S.: Bounded polymorphism in session types. Technical Report TR-2003-132, Department of Computing Science, University of Glasgow (2003)

[HKT94] Honda, K., Kubo, M., Takeuchi, K.: An interaction-based language and its typing system. In: Halatsis, C., Philokyprou, G., Maritsas, D., Theodoridis, S. (eds.) PARLE 1994. LNCS, vol. 817, pp. 398–413. Springer, Heidelberg (1994)

[Hon93] Honda, K.: Types for dyadic interaction. In: Best, E. (ed.) CONCUR 1993. LNCS, vol. 715, pp. 509–523. Springer, Heidelberg (1993)

[HVK98] Honda, K., Vasconcelos, V., Kubo, M.: Language primitives and type discipline for structured communication-based programming. In: Hankin, C. (ed.) ESOP 1998 and ETAPS 1998. LNCS, vol. 1381, pp. 122–138. Springer, Heidelberg (1998)

[HYC08] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In: POPL 2008 (to appear, 2008)

[NT04] Neubauer, M., Thiemann, P.: Session types for asynchronous communication. Universität Freiburg (2004)

[VRG04] Vasconcelos, V., Ravara, A., Gay, S.: Session types for functional multithreading. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 497–511. Springer, Heidelberg (2004)

[VVR03] Antonio Vallecillo, Vasco Vasconcelos, and António Ravara. Typing the behavior of objects and component using session types. ENTCS, 68(3) (2003)

[Yos07] Yoshida, N.: Personal communication (September 5, 2007)

[YV06] Yoshida, N., Vasconcelos, V.: Language primitives and type disciplines for structured communication-based programming revisited. In: SecRet 2006, ENTCS. Elsevier, Amsterdam (2006)